FAX TRANSMITTAL COVER SHEET

CONLEY ROSE, P.C.
600 Travis, Suite 7100
Houston, Texas 77002
Fax Number: (713) 238-8008
Telephone Number: (713) 238-8000

ORIGINAL WILL FOLLOW VIA:
- ☐ MAIL
- ☐ INTERNATIONAL AIRMAIL
- ☐ COURIER
- ☒ WILL NOT FOLLOW
- ☐ HAND DELIVERY
- ☐ WITH ENCLOSURE(S)
- ☐ WITHOUT ENCLOSURE(S)

PLEASE DELIVER THE FOLLOWING PAGES IMMEDIATELY TO:

NAME: **MAIL STOP APPEAL BRIEF - PATENTS**

FIRM: **U.S. PATENT & TRADEMARK OFFICE**

CITY: **ALEXANDRIA, VIRGINIA**

FAX NO: **(703) 872-9306**

RECEIVED
CENTRAL FAX CENTER
FEB 14 2005

REMARKS: **Serial No. 09/540,952, filed 03/31/2000**
Attached hereto is an Appeal Brief for filing with the U.S. Patent and Trademark Office. Please acknowledge receipt of this facsimile transmission.

Total Number of Pages (Including This One): **TWENTY-EIGHT (28)**

FROM: Jonathan M. Harris, Direct Dial No. 713/238-8045

DATE: February 14, 2005 CLIENT/MATTER NO. 2003017032-1 (2162-15100)

IF YOU DO NOT RECEIVE ALL THE PAGES,
PLEASE CALL BACK AS SOON AS POSSIBLE.

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

ORIGINAL

PATENT APPLICATION

ATTORNEY DOCKET NO. 200301702-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s):	Carl A. WALDSPURGER et al.	**Confirmation No.:**	1928
Application No.:	09/540,952	**Examiner:**	C. O. Kendall
Filing Date:	03/31/2000	**Group Art Unit:**	2122
Title:	EFFICIENT, TRANSPARENT, AND FLEXIBLE LATENCY SAMPLING		

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Sir:

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on 01/10/2005.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) $500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

( ) (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d) for the total number of months checked below:

( )	one month	$120.00
( )	two months	$450.00
( )	three months	$1020.00
( )	four months	$1590.00

( ) The extension fee has already been filled in this application.

(X) (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of $500.00. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

( ) I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450. Date of Deposit:_____

OR

(X) I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (703) 872-9306 on 02/14/2005

Number of pages: 27

Typed Name: Colleen F. Brown

Signature: Colleen F. Brown

Respectfully submitted,

Carl A. WALDSPURGER et al.

By _____

Jonathan M. Harris

Attorney/Agent for Applicant(s)
Reg. No. 44,144

Date: 02/14/2005

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellants:	Carl A. Waldspurger et al.	§	Confirmation No.:	1926
Serial No.:	09/540,952	§	Group Art Unit:	2122
Filed:	03/31/2000	§	Examiner:	C. O. Kendall
For:	Efficient, Transparent, and Flexible Latency Sampling	§	Docket No.:	200301702-1

APPEAL BRIEF

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: February 14, 2005

Sir:

Appellants hereby submit this Appeal Brief in connection with the above-identified application. A Notice of Appeal was filed via facsimile on January 10, 2005.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company (HPDC), a Texas Limited Partnership, having its principal place of business in Houston, Texas, through its merger with Compaq Computer Corporation (CCC) which owned Compaq Information Technologies Group, L.P. (CITG). The assignment from the Inventors to CCC was recorded on July 12, 2000, at Reel/Frame 010926/0351. The assignment from CCC to CITG was recorded on December 18, 2001, at Reel/Frame 012370/0533. The Change of Name document was recorded on December 2, 2003, at Reel/Frame 014177/0428.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF THE CLAIMS

Originally filed claims:	1-39.
Claim cancellations:	5, 18 and 31.
Added claims:	None.
Presently pending claims:	1-4, 6-17, 19-30 and 32-39.
Presently appealed claims:	1-4, 6-17, 19-30 and 32-39.

IV. STATUS OF THE AMENDMENTS

No claims were amended after the final Office action dated November 9, 2004.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The summary is set forth in the following exemplary embodiments that correspond to claims involved in the appeal. Discussions about elements and recitations of these claims can be found at least at the cited locations in the specification and drawings.

In accordance with one embodiment, as disclosed, for example in Appellants' disclosure pages 41-50 and Figures 13-15, a method of monitoring the performance of a program being executed on a computer system comprises executing the program on a computer system. The program has object code instructions. At intervals, the method comprises interrupting execution of the program (see e.g. Figure 3 and page 9), including delivering a first interrupt, and in response to at least a subset of the first interrupts, measuring a latency of execution of a particular object code instruction and storing the latency in a first database in such a way that the program remains unmodified.

In accordance with another embodiment, a computer program product, pages 5-6, is disclosed for sampling latency of a computer program having object code instructions. The computer program product is for use in conjunction with a computer system (Figures 1 and 12). The latency is sampled, without modifying the computer program, while the object code instructions are executing. The computer program product comprises a computer readable storage medium 24 on which a computer program mechanism is embedded. The computer program mechanism comprises one or more instructions to deliver interrupts at intervals during execution of the program. Pages 5-6. This act may include delivering a first interrupt. The computer program mechanism also comprises one or more instructions to measure a latency of execution of a particular object code instruction and one or more instructions to, in response to at least a subset of the first interrupts, store the latency value for the particular object code instruction in a first database. Pages 41-50 and Figures 13-15.

In accordance with yet another embodiment, a computer system (Figures 1 and 12) comprises a processor 22 for executing instructions and a memory 24 that stores instructions. The instructions comprise one or more instructions to

deliver interrupts at intervals during execution of the program (including delivering a first interrupt) to measure a latency of execution of a particular object code instruction, and, in response to at least a subset of the first interrupts, to store the latency value for the particular object code instruction in a first database. Pages 41-50 and Figures 13-15.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-4, 6-10, 14-17, 19-23, 27-30, 32-36, 38 and 39 are anticipated by Levine et al. (U.S. Pat. No. 6,134,710).

Whether claims 11-13, 24-26 and 37-39 are unpatentable over Levine et al. (U.S. Pat. No. 6, 134,710) in view of Krishnaswamy (U.S. Pat. No. 6,308,318).

VII. ARGUMENT

A. Overview of Levine

Levine is directed to a method that attempts to minimize the effect of long cache misses. The Abstract of Levine explains that Levine's contribution is directed to

> an automated method of tuning application programs to execute more efficiently. Based on several system parameters provided by the user, the disclosed method comprises profiling an application to determine where significant delays are occurring that result from long cache misses, constructing effective address tables to identify the effective addresses associated with the most significant delays, optimizing the placement of preload or touch instructions that initiate execution of identified instructions prior to their placement in the program sequence, building an optimized change file, and applying the optimized change file to the object code. The optimized change file may be inserted into the object code on a real-time basis.

The Examiner seems to have mainly focused on the text of column 8 of Levine. In that column, Levine states that

> [t]he events to be monitored by the performance monitor 80 are selected by the event detection and control logic 170 under control of MMCR0 110 and MMCR1 120. An accurate time base 190, and a thresholder 180 that may be loaded from a control field of MMCR0 110 are also depicted. The events to be monitored by the performance monitor 80 are implementation dependent and may be performance parameters such as the number of execution unit stalls and duration, execution unit idle time, memory access time, etc. The monitor mode control registers MMCR0 110 and MMCR1 120 control the operation of the performance monitor counters PMC0 130, PMC1 140, PMC2 150, through PMC7 160.

Col. 8, lines 4-11.

> Bits 10-15 of MMCR0 110 are used to store a software selectable threshold value (X), which enables a count when the threshold value is exceeded. The threshold value is exceeded when a decrementer with an initial value that equals the threshold value reaches zero before a selected event is completed. The threshold value is not exceeded when the selected event is completed before the decrementer, having an initial value that equals the threshold value, reaches zero. Bits 19-25 of MMCR0 110 are used to select the events to be monitored by PMC0 and bits 26-31 of MMCR0 110 are

> used to select the events to be monitored by PMC1. Similarly, MMCR1 120 bits 0-4 control the event selection for PMC2, bits 5-9 control event selection for PMC3, bits 10-14 control event selection for PMC4, bits 15-19 control event selection for PMC5, bits 20-24 control event selection for PMC6, and bits 25-28 control event selection for PMC7. There may be less than or more than eight performance monitor counters. The number of performance monitor counters is implementation dependent.

Col. 8, lines 36-55.

Based on the foregoing, Levine simply determines whether an event (e.g., a cache miss) took longer than a threshold amount of time. A counter is incremented each time a particular event takes longer than the threshold amount of time. The counter thus tracks the number of times an event took too long to complete. Levine does not disclose determining exactly how long the event took, only counting the number of times the event took too long.

B. Claims 1, 6-8, and 11-13

Applicants select claim 1 as representative of this group of claims. Claim 1 requires "measuring a latency of execution of a particular object code instruction." The above-quoted passages from Levine do not disclose actually measuring the latency of execution of an instruction. Levine, instead, teaches that various events can be "monitored" to determine whether the latency of an instruction is greater than a threshold. Levine does not teach actually measuring the instruction's latency. Levine teaches starting to decrement a counter from an initial value towards 0. This counter either reaches 0 or is stopped before reaching 0 by a "selected event." Col. 8, lines 36-44. If the counter reaches 0, then the monitored selected event did not occur within the time counted by the counter. Ultimately, the selected event will no doubt occur, but after the counter reaches 0, and Levine is incapable of knowing how much longer the event continued to be in progress before completing.

Applicants offer the following example to help illustrate this patentable distinction. A cache miss from a load instruction might ultimately take 100 clock cycles to be resolved (i.e., to retrieve the target data from memory and populate the cache). Levine's counter might start counting down from a threshold value of,

for example, 75 upon initially determining that a cache miss has occurred. In this example, the counter would reach 0 before the cache miss event resolved itself. The point of this example is to illustrate that Levine is incapable of knowing that the cache miss took 100 clock cycles to resolve. All Levine can determine is that the cache miss took more than 75 clock cycles to resolve. Levine's system will never know whether the cache miss ultimately took 76 clock cycles, 100 clock cycles, or 176 clock cycles to resolve. This is patentably different than actually "measuring a latency of execution of a particular object code instruction."

Accordingly, Levine does not disclose actually measuring the latency of execution of the instruction. It is evidently sufficient for Levine's purposes simply to know whether an instruction takes more than a threshold amount of time to execute.

Claim 1 also requires "storing the latency in a first database." Even if Levine did teach measuring the latency of execution of an instruction, which it does not, Levine does not teach storing the latency in a database. The Examiner attempted to find teachings of this limitation in Levine at column 8, lines 34-36 and column 12, lines 53-57. Those passages disclose that the "state of the execution units is also saved on interrupt," that "[b]its 10-15 of MMCR0 110 are used store a software selectable threshold value (X)," and that "using the number of processor cycles that is equivalent to the threshold value, and by using the average number of processor cycles per instruction, the number of instructions equivalent to the threshold value may be determined." These passages do not all teach or even suggest storing a latency that has been measured for the execution of an instruction.

The Examiner also cited Krishnaswamy in an obviousness rejection of some of the dependent claims. Krishnaswamy does not satisfy the above-noted deficiencies of Levine. For at least these reasons, the Examiner erred in rejecting all of the claims in this group.

C. Claims 2-4 and 9-10

These claims depend from claim 1 and thus are allowable at least for the same reasons as claim 1. The Examiner erred in rejecting these claims for an

additional reason. Applicants select claim 2 as representative of this group of claims. Claim 2 requires "determining an initial value of a cycle counter," "determining a final value of the cycle counter," and "measuring the latency based on the initial value and the final value." Levine does not at all teach measuring an instruction's execution latency, much less doing so in the way required by claim 2. While Levine may disclose the use of counters, it does not disclose using counters as required by claim 2. Krishnaswamy is also deficient in this regard.

D. Claims 14, 16, 17, 19-21, and 24-26

Applicants select claim 14 as representative of this group of claims. Claim 14 requires "one or more instructions to measure a latency of execution of a particular object code instruction." As explained above, Levine does not disclose actually measuring the latency of execution of an instruction. Levine, instead, determining whether an event (e.g., a cache miss) took longer than a threshold amount of time.

Claim 14 also requires "one or more instructions to...store the latency...in a first database." As explained above, Levine does not teach storing a measured latency in a database.

Krishnaswamy does not satisfy the above-noted deficiencies of Levine. For at least these reasons, the Examiner erred in rejecting all of the claims in this group.

E. Claims 15, 22, and 23

These claims depend from claim 14 and thus are allowable at least for the same reasons as claim 14. The Examiner erred in rejecting these claims for an additional reason. Applicants select claim 15 as representative of this group of claims. Claim 15 requires instructions to "determine an initial value of a cycle counter," to "determine a final value of the cycle counter," and to "measure the latency based on the initial value and the final value." Levine does not at all teach measuring an instruction's execution latency, much less doing so in the way required by claim 15. While Levine may disclose the use of counters, it does not disclose using counters as required by claim 15. Krishnaswamy is also deficient in this regard.

F. Claims 27, 29, 32-34, and 37-39

Applicants select claim 27 as representative of this group of claims. Claim 27 requires memory that includes "one or more instructions to measure a latency of execution of a particular object code instruction." As explained above, Levine does not disclose actually measuring the latency of execution of an instruction. Levine, instead, determines whether an event (e.g., a cache miss) took longer than a threshold amount of time.

Claim 27 also requires "one or more instructions to...store the latency...in a first database." As explained above, Levine does not teach storing a measured latency in a database.

Krishnaswamy does not satisfy the above-noted deficiencies of Levine. For at least these reasons, the Examiner erred in rejecting all of the claims in this group.

G. Claims 28, 30, and 35

These claims depend from claim 27 and thus are allowable at least for the same reasons as claim 27. The Examiner erred in rejecting these claims for an additional reason. Applicants select claim 28 as representative of this group of claims. Claim 28 requires instructions to "determine an initial value of a cycle counter," to "determine a final value of the cycle counter," and to "measure the latency based on the initial value and the final value." Levine does not at all teach measuring an instruction's execution latency, much less doing so in the way required by claim 28. While Levine may disclose the use of counters, it does not disclose using counters as required by claim 28. Krishnaswamy is also deficient in this regard.

VIII. CONCLUSION

For the reasons stated above, Appellants respectfully submit that the Examiner erred in rejecting all pending claims. It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such

extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's Deposit Account No. 08-2025.

Respectfully submitted,

Jonathan M. Harris
PTO Reg. No. 44,144
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANTS

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

IX. CLAIMS APPENDIX

1. (Previously presented) A method of monitoring the performance of a program being executed on a computer system, comprising:

executing the program on a computer system, the program having object code instructions;

at intervals interrupting execution of the program, including delivering a first interrupt; and

in response to at least a subset of the first interrupts, measuring a latency of execution of a particular object code instruction, storing the latency in a first database, the particular object code instruction being executed by the computer such that the program remains unmodified.

2. (Previously presented) The method of claim 1 wherein measuring the latency includes:

determining an initial value of a cycle counter;

performing the particular object code instruction;

determining a final value of the cycle counter; and

measuring the latency based on the initial value and the final value.

3. (Original) The method of claim 2 further comprising:

executing at least one instruction selected from the set consisting of (A) an instruction for ensuring that the particular object code instruction is performed after the initial value of the cycle counter is determined, and (B) an instruction for ensuring that the particular object code instruction is performed before the final value of the cycle counter is determined.

4. (Original) The method of claim 2 further comprising:

applying an adjustment to the final value.

5. (Cancelled).

6. (Original) The method of claim 1 wherein the particular object code instruction has a variable execution time.

7. (Original) The method of claim 1 wherein the particular object code instruction is a memory access instruction.

8. (Original) The method of claim 1 wherein the computer system includes a plurality of memory units, each memory unit of the plurality of memory units having a different range of access times, and further comprising:

associating the particular object code instruction with a memory unit in accordance with the latency and the range of access times for the memory unit.

9. (Previously presented) The method of claim 1 wherein measuring the latency includes:

determining an initial value of a cycle counter;

executing a first dependent instruction to provide a predetermined execution order;

performing the particular object code instruction;

executing a second dependent instruction to provide the predetermined execution order;

determining a final value of the cycle counter; and

determining the latency based on the initial value and the final value.

10. (Original) The method of claim 9 wherein the first and second dependent instructions are memory barrier instructions.

11. (Previously presented) The method of claim 1 wherein measuring includes:

identifying at least one issue block of instructions; and

interpreting the instructions of the at least one issue block;

wherein said particular object code instruction is in the issue block.

12. (Original) The method of claim 11 wherein said interpreting emulates a machine language instruction set of the computer system.

13. (Original) The method of claim 11 wherein said interpreting updates a state of the interrupted program as though each interpreted instruction had been directly executed by the computer system.

14. (Previously presented) A computer program product for sampling latency of a computer program having object code instructions while the object code instructions are executing without modifying the computer program, the computer program product for use in conjunction with a computer system, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

one or more instructions to deliver interrupts at intervals during execution of the program, including delivering a first interrupt;

one or more instructions to measure a latency of execution of a particular object code instruction; and

one or more instructions to, in response to at least a subset of the first interrupts, store the latency value for the particular object code instruction in a first database.

15. (Previously presented) The computer program product of claim 14 wherein said one or more instructions to measure the latency value include instructions to:

determine an initial value of a cycle counter;

perform the particular object code instruction;

determine a final value of the cycle counter; and

measure the latency based on the initial value and the final value.

16. (Original) The computer program product of claim 14 further comprising at least one instruction selected from the set consisting of (A) an instruction for ensuring that the particular object code instruction is performed after the initial value of the cycle counter is determined, and (B) an instruction for ensuring that the particular object code instruction is performed before the final value of the cycle counter is determined.

17. (Original) The computer program product of claim 15 further comprising one or more instructions to apply an adjustment to the final value.

18. (Cancelled).

19. (Original) The computer program product of claim 14 wherein the particular object code instruction has a variable execution time.

20. (Original) The computer program product of claim 14 wherein the particular object code instruction is a memory access instruction.

21. (Original) The computer program product of claim 14 wherein the computer system includes a plurality of memory units, each memory unit of the plurality of memory units having a different range of access times, and further comprising one or more instructions that associate the particular object code instruction with a memory unit in accordance with the latency value and the range of access times for the memory unit.

22. (Previously presented) The computer program product of claim 14 wherein said one or more instructions to measure the latency value include:

one or more instructions to determine an initial value of a cycle counter;

a first dependent instruction to provide a predetermined execution order;

the particular object code instruction;

a second dependent instruction to provide the predetermined execution order;

one or more instructions to determine a final value of the cycle counter; and

one or more instructions to measure the latency value based on the initial value and the final value.

23. (Original) The computer program product of claim 22 wherein the first and second dependent instructions are memory barrier instructions.

24. (Previously presented) The computer program product of claim 14 wherein said instructions to measure include:

one or more instructions that identify at least one issue block of instructions; and

an interpreter to interpret the instructions of the at least one issue block;

wherein said particular object code instruction is in the issue block.

25. (Original) The computer program product of claim 24 wherein the interpreter emulates a machine language instruction set of the computer system.

26. (Original) The computer program product of claim 24 wherein the interpreter updates a state of the interrupted program as though each interpreted instruction had been directly executed by the computer system.

27. (Previously presented) A computer system comprising:

a processor for executing instructions; and

a memory storing instructions including:

one or more instructions to deliver interrupts at intervals during execution of the program, including delivering a first interrupt;

one or more instructions to measure a latency of execution of a particular object code instruction; and

one or more instructions to, in response to at least a subset of the first interrupts, store the latency value for the particular object code instruction in a first database.

28. (Previously presented) The computer system of claim 27 wherein said one or more instructions to measure the latency value include instructions to:

determine an initial value of a cycle counter;

perform the particular object code instruction;

determine a final value of the cycle counter; and

measure the latency based on the initial value and the final value.

29. (Original) The computer system of claim 27 wherein the memory further comprises at least one instruction selected from the set consisting of (A) an instruction for ensuring that the particular object code instruction is performed after the initial value of the cycle counter is determined, and (B) an instruction for ensuring that the particular object code instruction is performed before the final value of the cycle counter is determined.

30. (Original) The computer system of claim 28 wherein the memory further comprises one or more instructions to apply an adjustment to the final value.

31. (Cancelled).

32. (Original) The computer system of claim 27 wherein the particular object code instruction has a variable execution time.

33. (Original) The computer system of claim 27 wherein the particular object code instruction is a memory access instruction.

34. (Original) The computer system of claim 27 further comprising:

a plurality of memory units, each memory unit of the plurality of memory units having a different range of access times, and

wherein the memory further comprises one or more instructions that associate the particular object code instruction with a memory unit in accordance with the latency value and the range of access times for the memory unit.

35. (Previously presented) The computer system of claim 27 wherein said one or more instructions to measure the latency value include:

one or more instructions to determine an initial value of a cycle counter;

a first dependent instruction to provide a predetermined execution order;

the particular object code instruction;

a second dependent instruction to provide the predetermined execution order;

one or more instructions to determine a final value of the cycle counter; and

one or more instructions to measure the latency value based on the initial value and the final value.

36. (Original) The computer system of claim 36 wherein the first and second dependent instructions are memory barrier instructions.

37. (Previously presented) The computer system of claim 27 wherein said instructions to measure include:

one or more instructions that identify at least one issue block of instructions; and

an interpreter to interpret the instructions of the at least one issue block;

wherein said particular object code instruction is in the issue block.

38. (Original) The computer system of claim 37 wherein the interpreter emulates a machine language instruction set of the computer system.

39. (Original) The computer system of claim 37 wherein the interpreter updates a state of the interrupted program as though each interpreted instruction had been directly executed by the computer system.

X. EVIDENCE APPENDIX

N/A

XI. RELATED PROCEEDINGS APPENDIX

None.